

MODELE

---

**DJANGO 02**

## CZYM W OGÓLE TAK NAPRAWDĘ SĄ MODELE

- ▶ Wspomniałem na poprzednim wykładzie, że w modelach dodajemy do naszej strony poszczególne elementy. To pewne uproszczenie
- ▶ DEFINICJA - model to typ obiektu, który reprezentuje dane i relacje w bazie danych
- ▶ Tak naprawdę głównym zadaniem modeli jest dodawanie, modyfikowanie oraz usuwanie danych bez konieczności pisania kodu SQL!

## KROK 1

- ▶ Utworzenie klasy dla modelu
- ▶ W odróżnieniu od czystego pythona przy definicji klasy nie używamy self - nie odnosi się do siebie, lecz do zewnętrznej biblioteki - models.
- ▶ Wybór elementów, które będziemy używać - w pierwszej kolejności konstruowanie prostych modeli kojarzy się z tworzeniem formularza. Czy elementów prostej www

## KROK 1 – PRZYKŁAD

```
# models.py
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=100)
    text = models.TextField()
    author = models.ForeignKey(User, on_delete=models.CASCADE)
```

- ▶ Klasyczny import
- ▶ W klasie znajdują się elementy. Można zakończyć zwykłym () ale nie ustalimy wówczas żadnych ograniczeń
- ▶ Coś takiego będzie trzeba zmigrować do bazy danych.
- ▶ Ale wpierw jeszcze zarejestrujemy nasz model w Django

## KROK 2 – DODANIE MODELU DO APLIKACJI, W ZASADZIE APLIKACJI DO DJANGO

- ▶ Trzeba dodać każdą aplikację do listy zainstalowanych - `INSTALLED_APPS` w pliku `settings.py`
- ▶ Może to być zwykły wpis jako nazwa - `'nazwa_aplikacji'` - django samo tworzy pliki konfiguracyjne
- ▶ A można od razu pełną konfigurację - `'nazwa.apps.NazwaConfig'`

## SKĄD WZIĄĆ MODELE – ELEMENTY

- ▶ Większość jest dość intuicyjna (przynajmniej po 15 latach z php/html itp)
- ▶ Oczywiście najwłaściwszym byłoby przeczytać, mieć pod ręką dokumentację biblioteki. Pomagają różne czaty.
- ▶ Część rzeczy naprawdę się zapamiętuje, ale trochę przykładów podam

## PRZYKŁADY PÓL W MODELU

- ▶ CharField - pole tekstowe - jak zaczniemy wpisywać parametr np max\_length - to sam podpowie
- ▶ IntegerField - pole na liczby
- ▶ DateTimeField - data i czas - można precyzyjnie format ustalać
- ▶ I wiele wiele innych. Zasada użycia - nazwa\_pola = models.TypPola(parametry)

## PRZYKŁADY PÓL W MODELU

- ▶ ForeignKey służy do tworzenia relacji „wiele do jednego” między modelami w Django. Aby użyć pola ForeignKey, musisz podać dwa argumenty pozycyjne: model, do którego odwołuje się pole, i on\_delete, który określa, co zrobić z obiektami powiązanymi, gdy obiekt nadrzędny zostanie usunięty

```
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=50)

class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.ForeignKey(Author, on_delete=models.CASCADE) # usuwanie obiektu
    książki po usunięciu obiektu autora
```



## CZYM SĄ WIDOKI I JAK JE POWIĄZAĆ Z MODELAMI

- ▶ Views w Django to funkcje lub klasy, które odpowiadają za obsługę żądań HTTP i wyświetlanie odpowiedzi HTTP. Views są związane z modelami przez to, że mogą odbierać dane z modelu jako parametry lub jako atrybuty obiektu widoku

```
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=50)

class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.ForeignKey(Author, on_delete=models.CASCADE) # usuwanie obiektu
    książki po usunięciu obiektu autora
```

# CZYM SĄ WIDOKI I JAK JE POWIĄZAĆ Z MODELAMI

- ▶ Można powiedzieć, że views określa zachowania i interakcje aplikacji webowej

```
from django.http import HttpResponse
from .models import Book

def book_list(request):
    # pobieramy wszystkie książki z bazy danych
    books = Book.objects.all()
    # tworzymy listę książek do przekazania do szablonu
    book_list = books.values_list('title', 'author', 'price')
    # wracamy do szablonu book_list.html
    return render(request, 'book_list.html', {'book_list': book_list})
```

```
def post_list(request):
    return render(request, 'blog/post_list.html', {})
```

# PRZYPOMINAM O URLS

- ▶ W URLS dodajemy dowiązania do naszych aplikacji. W dużym skrócie. Jest to ściśle powiązane z Views. URLS jest miejscem, gdzie możemy ustawić mapowania url do widoku. A w views ustawiamy co ma się uruchomić po skorzystaniu z url.

projekt

```
1
2 from django.contrib import admin
3 from django.urls import path, include
4 #from pacjenci.views import test_response
5 urlpatterns = [
6     path('admin/', admin.site.urls),
7     path('pacjenci/', include('pacjenci.urls')) #to prowadzi do urls w aplikacji
8 ]
9
```

```
1 from django.urls import path
2 from pacjenci.views import spis_pacjentow
3 urlpatterns = [
4
5     path('pacjenci/', spis_pacjentow)
6 ]
7
```

- ▶ W dużym uproszczeniu - odwiedzenie ścieżki pacjenci wywołuje funkcję spis\_pacjentów

# SKORO WIDOK ZWRACA, CO MA SIĘ POJAWIĆ PO SKORZYSTANIU Z URL

```
1 from django.shortcuts import render
2 from django.http import HttpResponse
3
4 def test_response(request):
5     return HttpResponse("To jest nasz pierwszy test")
6
```

```
1 from django.contrib import admin
2 from django.urls import path
3 from filmyweb.views import test_response
4
5 urlpatterns = [
6     path('admin/', admin.site.urls),
7     path('test/', test_response)
8 ]
9
```



WIDOKI SĄ W SUMIE BEZ SENSU JAK NIE PODŁĄCZYMY SOBIE ŻADNEGO SZABLONU...

## CZYLI WŁĄCZAMY DO TEGO WSZYSTKIEGO TEMPLATES

The screenshot shows the PyCharm IDE interface. On the left, the 'Project' sidebar displays the file structure of a Django project named 'kurs\_udemy'. The 'templates' directory is highlighted with a red rectangle, and the file 'pacjenci.html' is selected within it. The main editor window shows the content of 'pacjenci.html', which is a Django template. The template includes a doctype declaration, HTML lang attribute, head section with meta tags for charset, viewport, and http-equiv, and a title 'Document'. The body section contains an h2 tag with the text 'Znajdować się tutaj będzie baza pacjentów' and a Django loop for displaying patient names. The loop is commented with '#dodane dynamiczne przekazywanie pojedynczych danych - taka niewielka pętla'.

```
1 <!doctype html>
2 <html lang="pl">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport"
6         content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
7     <meta http-equiv="X-UA-Compatible" content="ie=edge">
8     <title>Document</title>
9 </head>
10 <body>
11 <h2>Znajdować się tutaj będzie baza pacjentów</h2>
12
13
14 {% for imiona_pacjentów in imiona_pacjentów%}
15 <p>{{ imiona_pacjentów.Nazwisko }}</p>
16 {% endfor %}
17 #dodane dynamiczne przekazywanie pojedynczych danych - taka niewielka pętla
18 </body>
19 </html>
20
21
22
```

WIDOKI SĄ W SUMIE BEZ SENSU JAK NIE PODŁĄCZYMY SOBIE ŻADNEGO SZABLONU...

---

## CZYLI WŁĄCZAMY DO TEGO WSZYSTKIEGO TEMPLATES

Zasadnicze pytanie, czy projekt w ogóle widzi nasze szablony. Jak korzystamy ze wspomagania w Pycharm Pro, to z automatu jest to w ustawieniach. W razie W trzeba znaleźć w settings odpowiednią linię i dodać wpis

```
54
55     TEMPLATES = [
56         {
57             'BACKEND': 'django.template.backends.django.DjangoTemplates',
58             'DIRS': [BASE_DIR / 'templates']
59         },
60         'APP_DIRS': True,
61         'OPTIONS': {
62             'context_processors': [
63                 'django.template.context_processors.debug',
64                 'django.template.context_processors.request',
65                 'django.contrib.auth.context_processors.auth',
66                 'django.contrib.messages.context_processors.messages',
67             ],
```

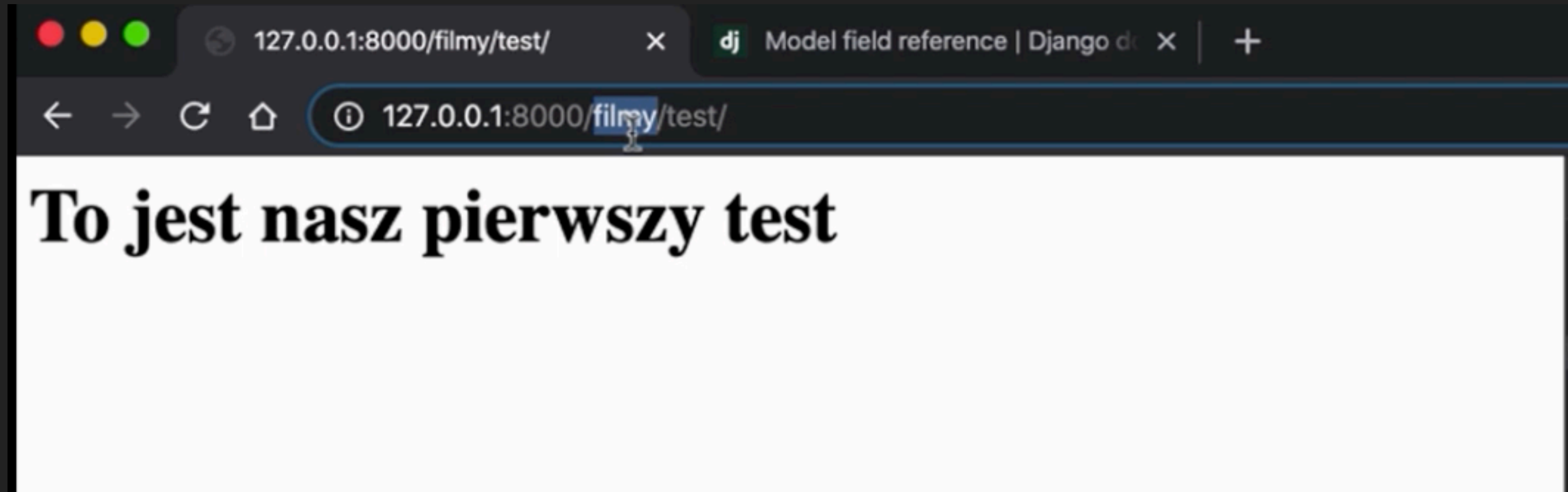
WIDOKI SĄ W SUMIE BEZ SENSU JAK NIE PODŁĄCZYMY SOBIE ŻADNEGO SZABLONU...

---

## CZYLI WŁĄCZAMY DO TEGO WSZYSTKIEGO TEMPLATES

Dobrze wcześniej stworzyć urls dla aplikacji osobno. Przykład już był.

Później łatwiej zarządzać aplikacjami / widokami



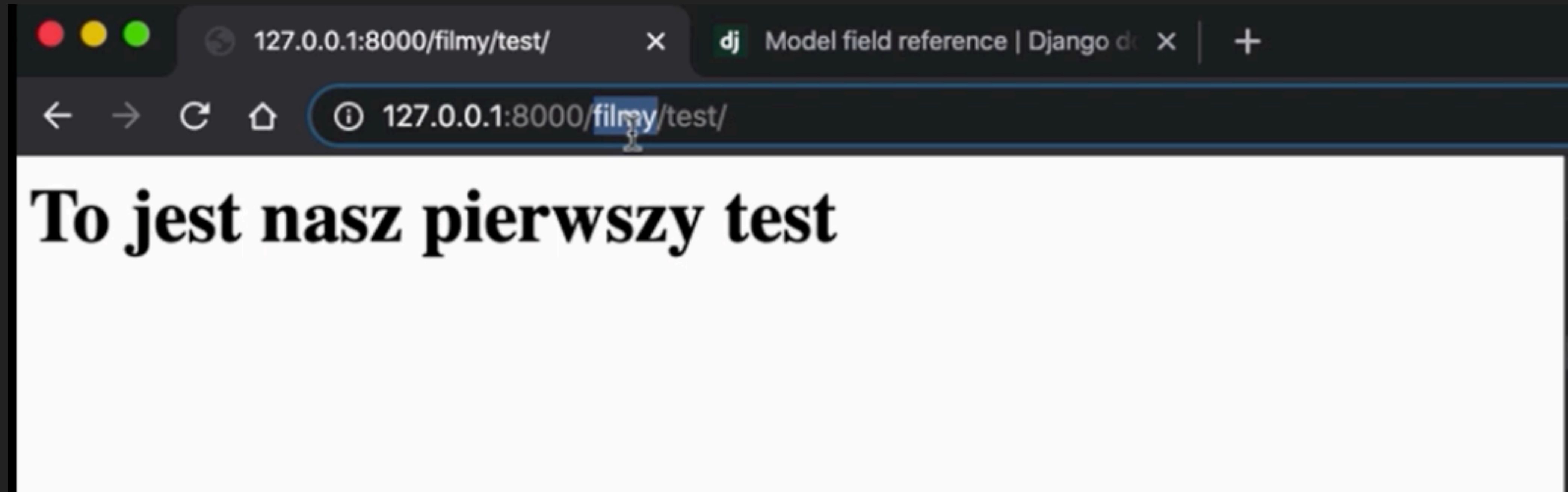
WIDOKI SĄ W SUMIE BEZ SENSU JAK NIE PODŁĄCZYMY SOBIE ŻADNEGO SZABLONU...

---

## CZYLI WŁĄCZAMY DO TEGO WSZYSTKIEGO TEMPLATES

Dobrze zawczasu stworzyć urls dla aplikacji osobno. Przykład już był.

Później łatwiej zarządzać aplikacjami / widokami





WIDOKI SĄ W SUMIE BEZ SENSU JAK NIE PODŁĄCZYMY SOBIE ŻADNEGO SZABLONU...

## CZYLI WŁĄCZAMY DO TEGO WSZYSTKIEGO TEMPLATES

W folderze Templates tworzymy sobie nasze szalony do widoków, w formacie html:

The image shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files like admin.py, apps.py, models.py, tests.py, urls.py, and views.py. A context menu is open over the file explorer, showing options like New, Cut, Copy, Copy Path/Reference..., File, Scratch File, Directory, Python Package, Python File, Jupyter Notebook, and HTML File (highlighted with a red box). The code editor shows the following HTML code:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8
9 </body>
10 </html>
```

The code editor also shows a file explorer at the bottom with files like kurs\_udemy/urls.py, pacjenci/urls.py, models.py, admin.py, <> pacjenci.html, settings.py, and <> test2.html. The file <> test2.html is selected, and its content is displayed in the editor area, showing the text "Tu wciskamy TAB." and "W niektórych wersjach podpowiada się samo".

WIDOKI SĄ W SUMIE BEZ SENSU JAK NIE PODŁĄCZYMY SOBIE ŻADNEGO SZABLONU...

---

## CZYLI WŁĄCZAMY DO TEGO WSZYSTKIEGO TEMPLATES

Trochę na poważniej zabieramy się za views - oczywiście w aplikacji - podłączamy szablon

```
from django.shortcuts import render
from django.http import HttpResponse

def test_response(request):
    # return HttpResponse("<h1>To jest nasz pierwszy test</h1>")
    return render(request, 'filmy.html')
```

WIDOKI SĄ W SUMIE BEZ SENSU JAK NIE PODŁĄCZYMY SOBIE ŻADNEGO SZABLONU...

---

## MOŻE ZAWARTOŚĆ ZROBIĆ NIECO BARDZIEJ DYNAMICZNIE

Chodzi o przekazywanie danych z aplikacji do html. Najpierw widok - na razie w jednym pliku

```
from django.shortcuts import render
from django.http import HttpResponse

def wszystkie_filmy(request):
    test = "To jest cos we views"
    return render(request, 'filmy.html', {'text': test})
```

WIDOKI SĄ W SUMIE BEZ SENSU JAK NIE PODŁĄCZYMY SOBIE ŻADNEGO SZABLONU...

---

## MOŻE ZAWARTOŚĆ ZROBIĆ NIECO BARDZIEJ DYNAMICZNIE

Później podłączamy to do html. Do tego aby podłączyć taką właśnie dynamiczną treść używa się czegoś, co się nazywa syntaksami - `{{ text }}` - to co się znajduje w klamrach to jest dynamiczny content. To zostało dynamicznie z views przekazane

```
from django.shortcuts import render
from django.http import HttpResponse

def wszystkie_filmy(request):
    test = "To jest cos we views"
    return render(request, 'filmy.html', {'text': test})
```

```
<p>{{ film }}</p>
```



WIDOKI SĄ W SUMIE BEZ SENSU JAK NIE PODŁĄCZYMY SOBIE ŻADNEGO SZABLONU...

---

## MOŻE ZAWARTOŚĆ ZROBIĆ NIECO BARDZIEJ DYNAMICZNIE

W sumie aby wyświetlić nieco więcej warto zrobić coś na kształt pętli.

Pętla jakby sprawdzała czy coś się nie zmienia.

```
{% for imiona_pacjentów in imiona_pacjentów%}  
<p>{{ imiona_pacjentów.Nazwisko }}</p>  
{% endfor %}  
#dodane dynamiczne przekazywanie pojedynczych danych - taka niewielka pętla
```

Pętla dla każdej zmiennej tworzy też nowy paragraf - fajnie to ustawia

WIDOKI SĄ W SUMIE BEZ SENSU JAK NIE PODŁĄCZYMY SOBIE ŻADNEGO SZABLONU...

---

## MOŻE ZAWARTOŚĆ ZROBIĆ NIECO BARDZIEJ DYNAMICZNIE – JESZCZE BARDZIEJ – ORM

Możemy sobie zaimportować obiekty z naszej bazy danych

```
def spis_pacjentow(request):  
    wszyscy = Pacjent.objects.all()  
    return render(request, template_name: 'pacjenci.html', context: {'imiona_pacjentów': wszyscy}) #wyświetla obiekty z bazy danych
```

Tym razem mamy naprawdę dynamiczną zawartość